Mobile Agents - Smart Messages[1]

Leon Hurst, Pádraig Cunningham
Computer Science Department,
Trinity College Dublin,
Dublin 2, Ireland.

Fergal Somers,
Broadcom Éireann Ltd.
Kestral House, Clanwilliam Place,
Dublin 2, Ireland.

Abstract. Wireless communication with Mobile Computing devices is known to be problematic. It is very different in character from conventional communication over wired networks. Since many distributed applications make assumptions about network characteristics, they may not be used in a hostile mobile environment.

We are proposing a new kind of messaging system which incorporates adaptive behaviour *into the messages themselves*. We call these 'Smart Messages', and implement them using Mobile Agents. The metaphor we use is of a message being delivered by a courier (Mobile Agent), on a potentially unresolved route. The 'intelligence' is in the messages themselves rather than in the network.

The approach taken expands on the self-routing capabilities of current Mobile Agent systems such as Aglets or Telescript. We aim to provide structured support for handling the particular problems associated with wireless communications. These include very limited, variable and asymmetric bandwidth, frequent and prolonged disconnections, geographical mobility and high usage costs. 'Smart Messages' offer an *efficient*, *adaptable* and *robust* solution to many of these problems.

1 Introduction

The Mobile Computing environment of today is characterised by very limited, variable and asymmetric bandwidth, frequent and prolonged disconnections, geographical mobility, severe resource restrictions and complex data management issues [6, 11]. In addition to these familiar issues there is also the crucial element of *dynamism*. Agent technology has already been proposed [4] as a possible solution to many of these problems.

We advocate the use of Mobile Agents as a replacement for conventional data packets in Mobile Communications. These agents manage their own routing, recovery and filtering behaviour rather than relying on "smart" [7, 9, 17] or "dumb networks". The dynamism issue is addressed by allowing these 'Smart Messages' to 'stay alive' until transmission time, when they are suspended and serialised. When 'alive' they can receive information regarding local events and *adapt to them dynamically*. This is the crux of our approach. This behaviour is encoded in State Machines with the ability to survive transmission between machines. Actions can be effected through named functions, supplied functions and external services.

1.1 Related work

Much work has been done in the area of Mobile Computing & Communications. The following paragraphs discuss some related work.

---

[1] The authors can be contacted at `leon.hurst@cs.tcd.ie`. This work is funded by Broadcom Éireann Ltd.

Rover and WitII are toolkits for building mobile applications. Rover [12] uses queued RPC and Relocatable Dynamic Objects to maximise use of the wireless communications medium at an application partitioning level. WitII [18] uses intelligent caching and pre-fetching based on hyper-object information.

Work at Columbia University [1, 7] proposes the creation of Proxy Servers (on the edge of the wired network) and the placement of message filters at these proxies. Messages sent to a Mobile Terminal are passed through these filters before they are dispatched over the hostile wireless connection to the mobile. A currently unresolved limitation of the proxy server is its inability to dynamically accept filters. This might adversely affect its ability to adapt to change as it cannot update its filters.

Work on Agent Tcl [13, 14] approaches the problem from an application partitioning point of view. IBM is engaged in research into Mobile Computing [5]. They take an Agent based approach where Intelligent Mobile Agents meet at Agent Meeting Points in order to acquire services. They use a Travel Reservation scenario to describe their system.

Finally the Telemedia Group, MIT, have been doing some work on incorporating interpreted code into IP packets [17, 19]. Code is piggy-backed onto the Options field of an IP packet. Such code might be executed at routers capable of interpreting the supplied code.

We have observed that a Mobile Terminal spends most of its time in a disconnected state. Hence outbound messages, usually requests, are queued at the Mobile Terminal until a connection is established [5, 12]. Furthermore inbound messages to a Mobile Host is typically *sent as a result* of requests made by the mobile. If the *originating requests can be filtered at the source* (given that they spend most of their life queued at the mobile anyway), then this would efficiently remove a significant amount of outbound traffic, inbound traffic and the associated server load needed to generate this traffic. The reason this works is due to the fact that such requests can become redundant when queued for too long. For instance map segments and traffic or weather information are only useful if they arrive on time.

To the best of our knowledge there is no system which empowers *transmitted messages* with the ability to dynamically adapt to environmental events, in a coherent, structured yet refinable manner.

In the remainder of this paper we present the context of our work (Section 2), a view of Mobile Agents as ‘Smart Messages’ routed by Agent Airports (Section 3), the dynamic adaptability of our ‘Smart Messages’ (Section 4), and conclude with a summation (Section 5).

2 The TNET system

TNET stands for “Tourist Network” [10]. The project proposes the creation of a country-wide online and interactive tourist network. The aim is to provide tourists with a single system through which they can access *all* relevant computerised services. The most interesting aspect from our point of view is that of the mobile tourist. Such a tourist might be touring the country in a coach, caravan or hire-car. Useful services for a tourist would include: Route Guidance, Traffic Service, Weather Service, Emergency Notification, Inter-tourist Communications and Online Hotel Booking.

The Network Architecture we have adopted is shown in Figure 1. The inner core represents a network of servers. The outer layer consists of a network of Mobile Support Stations which acts as a gateway between the Mobile Terminals and the rest of the world. Finally, Mobile Terminals exist beyond the wired network and connect intermittently to the MSS layer using GSM.

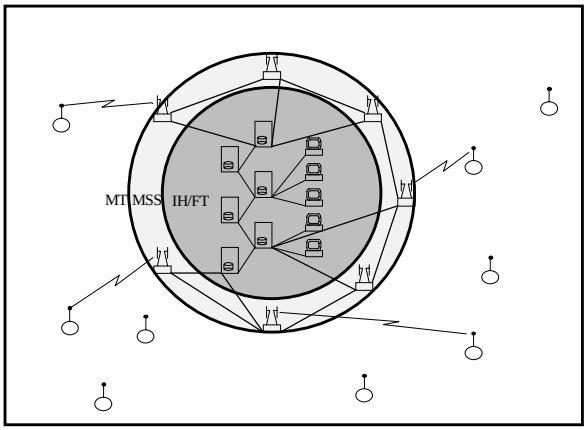

Figure 1. Network structure of **TNET**.

2.1 Limitations of Current Technologies

A Distributed Mobile System consists of several elements including the following:

- Communications
- Resource (data) management
- Application partitioning
- Colossal and increasing volumes of information

Communications

The communications issue is possibly the most prominent in a Mobile Computing environment, given its effect on other systems and the severe limitations which it incurs. Current communications protocols, such as streams and RPC, have been shown to be unsuitable [4]. The duration of communication must be short and packet-based, since disconnections are frequent. This makes streams unsuitable as they are long lived in nature. The volume of traffic is also important as our bandwidth is limited, hence RPC has been specifically identified as being inadequate [4]. These protocols are rather unintelligent with little or no recovery, defaulting to re-sends. Both streams and RPC have unfortunate failure characteristics which can leave both communicating entities in an inconsistent state.

There are many services which require one-to-many communication where the sender and the receiver are loosely coupled. We are essentially speaking of event based communication. Most current event systems are built for use on fixed networks, for example CMIP [2] or the new CORBA Event service [16]. These events are used for static entities (servers), are not queued, and are announced via centralised Event Managers. These solutions are used for tightly coupled, but geographically dispersed systems. Additionally, the range of events, their routing, and filtering are hard-coded

into the system and cannot be extended dynamically. What happens to events when Mobile Hosts (sending or receiving) are disconnected? Are they queued, re-routed or discarded? How do we intelligently handle event distribution, propagation and filtering?

In **TNET** we require a decentralised event system. It must allow the dynamic registration of events. Events must be queued for disconnected subscribers (Mobile Terminals), and when reconnected delivered to those subscribers regardless of their new location.

3 Mobile Agents and the Agent Airport

In this section we initially describe our solution to the problems described in the preceding section. This includes our Reference Model. Next, the structure of Mobile Agents and of Agent Airports is presented. Finally we explain the structure and operation of State Machines and their Processors. They are the means by which 'Smart Messages' are implemented. The operation of a State Machine occurs within the context of a host Agent Airport, and governs the routing, filtering, and recovery behaviour of the enclosing Mobile Agent.

3.1 Reference Model

A simplified Reference Model for our agent based framework is shown in Figure 2. We have identified four issues in Mobile Computing that have not been adequately resolved by current technologies and methodologies for very large Mobile Distributed Systems. In each case we offer a standardised, agent based service which addresses the associated problems.

- *Communications* - Agent Airport and Distributed Events.
- *Data Management* - Proxy Agents.
- *Application Partitioning* - AMA Agents[2].
- *Increasing Data Volumes* - RPA Agents.

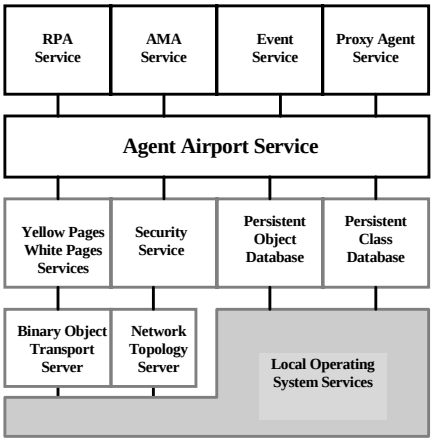

Figure 2. Reference Model.

[2] Abbreviations include AMA for Autonomous Mobile Agents and RPA for Record-Playback Agents.

The basic agent service which addresses the issue of communications directly is the Agent Airport and its 'Smart Messages'. The Agent Airport is the central metaphor and is useful in understanding the other services. These services will be the subject of later papers. However, it should be noted that the Distributed Events service is used to perform actions such as announcing Mobile Terminal reconnection.

3.2 Mobile Agents

A Mobile Agent has three main components. The first is a Passport which contains information about the Mobile Agent itself. This includes:

- Globally Unique Identifier (GUID) for itself,
- GUID for its creator Service Instance and Service Type[3],
- Date of Creation,
- Set of Visas governing access to secure resources.

The second, payload component, contains the application-level message. The last component is a set of Behavioural State Machines and State Machine Processors which allow it to perform its own filtering, routing and recovery, independent of the sending or receiving application. An empty messages with minimal behaviour will be of the order of 256 bytes in size.

The Passport is used primarily to identify the Mobile Agent to the host Agent Airport and its support services. It also includes the identity of its creator and its capabilities (Visas). A Visa is an authenticated stream of bits representing a public key, capability, or digital signature. The payload can be any transportable entity from a simple byte array or complex object hierarchy to an application task. Entities that may not be transported include local system resources (running GUIs or open file handles) and secure resources such as Visas and Agent Airport Services.

The behavioural State Machines are the component that distinguishes 'Smart Messages' from more inert messages found in conventional protocols like TCP/IP. They constitute a form of primitive Reactive Planning and are implemented as *persistent* State Machines which are triggered by events. The State Machines, which represent Goals and Sub-goals, are named and maintained in a pool. The pool can be added to as the result of processing a State Machine. This occurs when a Sub-goal in the current State Machine is encountered.

For each of the three tasks (filtering, routing and recovery) there is a State Machine Processor on which the State Machines execute (see Figure 3). The processors are akin to a conventional CPU, operating on State Machines rather than on subroutines. This gives the 'Smart Messages' a "fire-and-forget" behaviour. Their performance depends on how well the behavioural State Machines are exploited. See Section 4. The execution model is that of re-entrant finite state machines. Within each state of a state machine there is a block of code which is executed on entry to the state. Finally both the states and their enclosing state machines can be interrupted by events. The synchronisation of agents is facilitated by their behaviours and the Event system.

The Mobile Agent definition is dynamically extensible so that services and applications may specialise it for their own purposes. AMAs and RPAs, mentioned in

[3] The relationship between a Service Type and a Service Instance is similar to that of a Banking Group and an office of that Group at a particular address.

Section 2, are examples of such a specialisation. Instances of the resulting specialisation would be submitted to the Agent Airport in the same manner as the submission of basic Mobile Agents. Automatic distribution of newly specialised classes is catered for in our prototype.

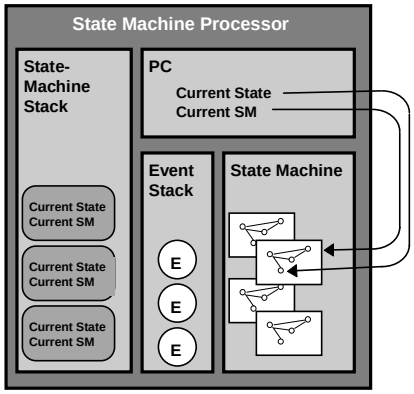

Figure 3. State Machine Processor (SMP)

3.3 Agent Airport

The structure of an Agent Airport is illustrated in Figure 4. The first role of the Airport is to accept Mobile Agents at the Check-In and to route them to the next Airport. Secondly, it receives Mobile Agents on its Inbound gates and hands them to the Arrivals. There they must choose a service or application to be delivered to. If no suitable service can be found, an alternative Airport must be chosen to which they might be transferred.

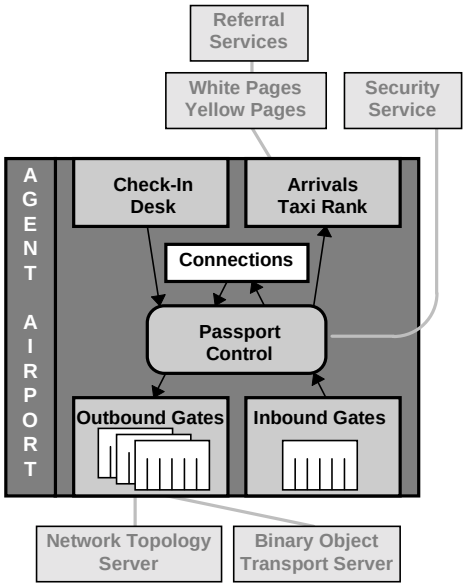

Figure 4. Agent Airport and associated services.

The two major points of interest in the Agent Airport are the Outbound gates and the Arrivals. We describe each in turn.

When Mobile Agents are passed to the Outbound gates they are handed a reference to the Agent Airport and their behavioural State Machines are (re)activated. The routing behaviour chooses a route and submits itself for transport to the first destination Agent Airport. It may use the Network Topology, Yellow and White pages services in choosing its route. It is then queued at the appropriate gate for transport. While queued a Mobile Agent may receive events which can trigger its filtering behaviour to kill, delay, resubmit or modify itself. An example might be a request that becomes outdated or redundant.

At the other end of the Airport we have Mobile Agents trying to contact destination service instances or applications. To do this they use a combination of the Yellow and White page services, and other referral services. If a service cannot be located the Mobile Agent has the option of routing to another Agent Airport that might support the service.

We handle four levels of routing in a hierarchical fashion: Content, Service, Service Instance and Agent Airport. A plan to route between two entities at one level will result in a route possibly involving several hops at a lower level. For example, requesting a routing between two service instances could result in a Agent Airport level route, which in turn could involve several hops. Finally, services and applications intending to be the destination of Mobile Agents have to implement a standard interface so that they may receive these agents from the Agent Airport as they arrive.

3.4 State Machines and their Processors

'Smart Messages' are responsible for their own filtering, routing and recovery. These are three orthogonal aspects of a 'Smart Message' which operate independent of each other. We will initially look at how a Reactive Planning system [15] is implemented using State Machines, and then examine some classes of events that affect the planning system as it executes.

Reactive Planning and State Machines

A Mobile Agent is alive and responsive to events at two particular times. The first is between the time it is submitted to an Agent Airport and the time it leaves that Airport. The second is from the time the Mobile Agent is received by a destination Airport and the time it leaves that Agent Airport.

Reactive Planning systems consist of Goals, Tasks and Actions. A State Machine represents a Goal to be achieved and consists of a network of States (Tasks) representing a well-structured plan to achieve that Goal (see Figure 5). States can be blocking, in which case the immediate processing of the State Machine is blocked pending an event matching the out transition arcs for that State. States contain Tasks to be performed which are Blocks of Actions. The first type of Action is a Primitive, that is an operation which can be directly called upon and results in success, failure or an error condition. The second type of Action is a Sub-goal. This Goal will cause a new State Machine to be created and popped onto the State Machine Stack. This goal subsumption leads to the creation of Goal hierarchies.

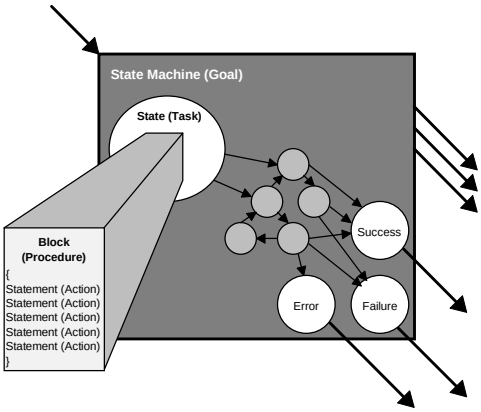

Figure 5. Hierarchy of State Machines representation of a Reactive Planning System.

The Reactive Planning system determines how interruptions to the normal deliberative execution of the State Machine Processor, are to be treated. External events must be reacted to. These external events can be directed to any of the State Machines on the stack and are processed like normal arc transitions. At the finest granularity they are handled in the current State of the State Machine. The State Machine as a whole can also subscribe to handle external events. If the State does not explicitly handle the incoming event, then it is passed to the State Machine. If that in turn does not handle the event, then the event propagates down through the State Machine stack till it arrives at the original Goal State Machine. This allows the Mobile Agent to specify reactive behaviour.

Some important aspects of this system are as follows:-

1. Higher Goals should be insulated from the propagation of errors generated by Sub-goals
2. Sub-goals (plans) can be supplied by the Agent Airport and its support services. This means that Mobile Agents do not have to have completely specified plan hierarchies bundled with them at dispatch time. The Mobile Agent can obtain plans on an “as needs” basis as it progresses through the path of Agent Airports.
3. Goals are modelled as State Machines which make transitions based on Goal-level events (for example `cost_too_high`), events which are the result of executing Primitive Actions, and on external events. The external events allow the planning system to react to incidents that occur outside the Mobile Agent.
4. Goals (State Machines) are linked hierarchically in a Goal–Sub-goal relationship.
5. Finally, a Procedure can either be interruptible or non-interruptible. This means that pending events will be examined after the execution of each Action in an interruptible Procedure.

The Procedure which implements a Task is represented by a Block. Blocks have Statements which are interpreted. These Statements can result in Primitive actions being invoked, such as `GoAgentAirport` and `GoServiceInstance`. Alternatively a Statement may result in the calling of a new State Machine (Sub-goal).

Classes of Events

During the time a Mobile Agent is activated it can listen for events which affects its planning. These events fall under several categories.

- *External events* which include events raised by entities outside of the Agent Airport. Typically this would be an application or service.
- *Timer events* are raised by the Agent Airport at a given interval.
- *Gate (queue) events* reflect a change in the nature of the Gate. For example '`time to next connection`' and '`number of agents ahead of you`'.
- *System events* announce changes in the operating conditions of the local host such as '`Battery Low`' warnings.
- *Transport events* announce the open/closed status of Gates.
- *Agent events* are directed at a specific agent and would include '`die`' and '`new resources`'.

The above events relate directly to the Mobile Agents and their behavioural State Machines. However their are three more events which are not received by Mobile Agents but relate to their transport status. These are `MA_XPORT_SUCCESS`, `MA_XPORT_FAIL(condition)`, and `MA_XPORT_ABORT(user data)`. The Agent Airport is normally the recipient of these events.

4 Mobile Agents - Smart Messages

In this section we construct a practical example of a 'Smart Message'. The example is taken from the **TNET** system. A tourist arrives in a new city and decides to look for accommodation. The tourist generates an Accommodation Enquiry using his Personal Tourist application. The application takes this request and deploys it in a 'Smart Message'. The first two parts of the 'Smart Message' contain the data shown in Table 2.

Table 2. Passport and Payload components of a 'Smart Message'

Passport	
Self	-randomly generated GUID-
Service Instance	Application instance GUID
Service	Application type GUID
Datestamp	13:00 17 June 1997
Visas	Transport capabilities; Tourist's personal digital signature.
Payload	
Type	"An Accommodation Enquiry"
Content	The Accommodation Enquiry

The intelligence of the Smart Message is encoded in the State Machines. The simplest of these is the recovery State Machine which handles exceptional circumstances at the host machine. Such events would include; Battery low, Disk full and Disk error. Two more sophisticated behaviours are described in the next sections.

4.1 Source Filtering

The generated request is specified as becoming redundant five hours after its creation and should kill itself. If the 'Smart Message' observes the queuing of another 'Smart Message', of the same type and origin as itself, it should retire itself as it is being superseded. The State Machine for this is shown in Figure 6.

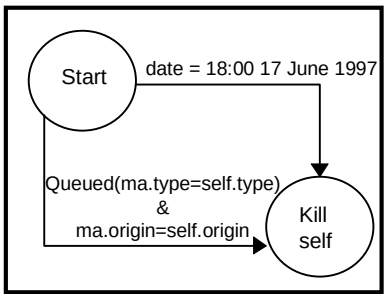

Figure 6. Filtering State Machine.

This State Machine handles some simple source filtering. It is termed "source" filtering as the behaviour is provided by the source application. It is also possible that the destinations of such 'Smart Messages' would like to kill them at their source. This is termed "destination" filtering and is implemented with Annihilator and Terminator Agents. It is beyond the scope of this paper to examine these.

4.2 Routing

Routing occurs at four different levels: Content, Service, Service Instance and Agent Airport. Each level will potentially have its own routing plan. It is often the case that a 'Smart Message' does not know its exact destination. It must therefore be learned from an Agent Airport or other Service. This is the case for Content and Service level routing which have no specified destination. Routing plans come from three sources:

1. Requests to the Agent Airport return routing plans devised by the Airport.
2. Requests to Services above the Agent Airport, known as Referral Services, result in plans devised by those Services.
3. Finally a routing plan can be supplied by the creating application or service. This would involve manual lookups of the local Yellow & White Page services.

It is also possible to have a mixture of the above methods. Hence one could try an application supplied routing plan. If it fails, default to an Agent Airport supplied route.

For this simple example we will use a fictitious Referral Service, named WiseMan, to find a Service Type which matches our Content Type. See Figure 7. This is done by invoking the `GetService` operation on the WiseMan referral service.

If WiseMan returns a Service Type we can request a routing plan from the local Agent Airport (AA prefix in Figure 7). This is done by invoking the `GoService` operation on the Agent Airport. If however, the WiseMan service cannot match a

Service to the specified Content Type, or the resulting Service is unreachable, we must use the basic Agent Airport referral services. We do this by invoking `GetService` on the Agent Airport instead of WiseMan.

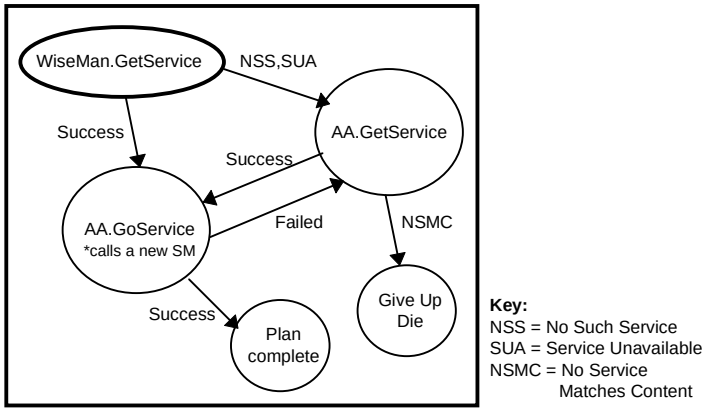

Figure 7. Simple Routing State Machine.

When a suitable Service is found the 'Smart Message' requests a routing plan from the Agent Airport (AA.GoService). This plan is supplied in the form of a set of new State Machines. Finally, the 'Smart Message' will either be routed to a Service Instance which can process its content, or it will run out of suitable Service Types that support such content.

5 Conclusions and Future Work

The advantages of 'Smart Messages' are as follows:

1. Filtering redundant requests queued to leave a Mobile Terminal.
2. Routing of 'Smart Messages' can be altered dynamically as the network changes.
3. A 'Smart Message' can be routed based on its Content Type or Service Type. In these cases it is the Agent Airport which constructs routing plans for the message.
4. 'Smart Messages' operate independently of the sending application, therefore removing any dependencies with the application.

We believe that the Agent Airport will have advantages over RPC under the following conditions:

1. When the Remote server contains large volumes of data to be processed;
2. There are hard real-time constraints;
3. Communication is costly;
4. The recipient of the mobile agent is itself located on another mobile terminal;
5. The processing capability of an Agent Airport Server is a limited.

Current prototyping is taking place using Smalltalk. Initial results are positive. However further work is required to fully evaluate the state machine model, to address the security issues including passports, and finally to determine the most common cases where the use of Smart Messages adds value.

References

[1] Athan A. and D. Duchamp, *Agent-Mediated Message Passing for Constrained Environments.* In Proceedings of the Mobile and Location-Independent Computing Symposium, August 1993.

[2] Black U., *Network Management Standards*, McGraw-Hill Series on Computer Communications, second edition, 1994.

[3] Borenstein N., *Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail.* In IFIP International Conference, June 1995, Spain.

[4] Chess D., C. Harrison and A. Kershenbaum, *Mobile Agents: Are they a good idea.* Technical Report, March 1995, IBM TJ. Watson Research Center, NY.

[5] Chess D., B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, *Itinerant Agents for Mobile Computing.* Technical Report, October 1995, IBM T.J. Watson Research Center, NY.

[6] Duchamp D., *Issues in Wireless Mobile Computing*, Internal Report, Computer Science Department, University of Columbia, NY.

[7] Duchamp D. and B. Zenel, *Intelligent Communication Filtering for Limited Bandwidth Environments.* In Proceedings of the 5th Workshop on Hot Topics in Operating Systems, IEEE, May 1995, Rosario WA.

[8] Gosling J. and H McGilton, *The Java Language Environment: A White Paper*. Sun Microsystems, 1995.

[9] Harrison G., *Smart Networks and Intelligent Agents.* In Mediacom'95, April 1995, Southampton, UK.

[10] Hurst L., *TNET: Executive Summary*. Internal Report available from `http://www.cs.tcd.ie/~lahurst/currentwork/execsum.html`

[11] Imielinski T. and B. Badrinath, *Wireless Computing*, In Communications of the ACM, October 1994/Vol.37, No.10.

[12] Joseph A., A. deLespinasse, J. Tauber, D. Gifford and M. Kaashoek. *Rover: A Toolkit for Mobile Information Access.* In Proceedings of the 5th Symposium on Operating Systems Principles, December 1995.

[13] Kotz D., R. Gray and D. Rus, *Transportable Agents Support Worldwide Applications.* In Proceedings of the 7th ACM SIGOPS European Workshop, September 1996, Connemara, Ireland.

[14] Kotz D., R. Gray, D. Rus, S. Nog and G. Cybenko, *Mobile Agents for Mobile Computing.* In Technical Report PCS-TR96-285, May 1996, Computer Science Department, Dartmouth College.

[15] Pryor L., *Adaptive Execution in Complex Dynamic Worlds.* Technical Report No. 53, 1994, Institute for Learning Sciences, North Western University, IL.

[16] Siegel J., *Common Object Services Specification, Volume I.* OMG Document No. 94-1-1

[17] Tennenhouse D. and D. Wetherall, *Towards an Active Network Architecture.* In Proc of Multimedia Computing and Networking, January 1996, CA.

[18] Watson T., *Efficient Wireless Communication Through Application Partitioning.* In Proceedings of the 5th Workshop on Hot Topics in Operating Systems, IEEE, May 1995, Rosario WA.

[19] Wetherall D. and D. Tennenhouse, *The ACTIVE Option.* In Proc of the 7th ACM SIGOPS European Workshop, September 1996, Connemara, Ireland.